International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Toward Intelligent Incident Response: A Framework for Self-Healing Production Systems

Pranav Gorak

Senior Site Reliability Engineer, IBM Organization

Abstract

The modern world of technology challenges production systems since they work in environments that are constantly evolving with influence from CI/CD and widespread use of the cloud. They call for fast setup and at the same time reliable and resilient operation. Even though looking into system behaviors has become much easier with observability and code-based infrastructure, organizations still rely on manual actions during incident response. As a result, there are risks for late deployment, uneven behavior, and more cases of long outages when the code is deployed frequently or the infrastructure becomes extremely unpredictable. Therefore, the study suggests a detailed approach to organize how self-healing production systems work. The framework brings in real-time data from telemetry and ties it to the way applications are deployed and setup using GitOps workflows. Enabling Kubernetes to control the system and deploying on different clouds allowed the system to find anomalies ahead of time, trigger corrective actions, and cut down mean time to repair (MTTR). Using telemetry and declarative methods, the framework helps developers manage the recovery of systems safe and quickly. Experimental data proves that adding intelligent incident response to CI/CD improves the system's stability, cuts risk at deployment time and increases the trust of both the development and operations teams. Thanks to automation of finding and resolving the common issues affecting infrastructure and applications, the platform moves production environments toward being more autonomous and steadier. In short, the results point to the fact that incorporating self-healing mechanisms is needed for reliability and at scale in the software systems used today.

Keywords: CI/CD, Self-Healing Systems, DevOps, Telemetry-Driven Deployment, Infrastructure Readiness, Kubernetes, GitOps, Observability, Incident Response

1. Introduction

Modern production environments have grown increasingly complex, driven by the adoption of cloud-native architectures, microservices, and high-frequency software releases. As a result, traditional approaches to incident response—often based on manual runbooks, reactive monitoring, or ad-hoc intervention—are no longer sufficient. Despite the use of infrastructure-as-code and sophisticated observability tools, many teams still address issues like deployment failures or infrastructure drift after they've already caused disruption. This reactive model delays recovery and exposes systems to avoidable risk. CI/CD pipelines have made deployments faster and more consistent, but they frequently lack awareness of the infrastructure conditions they depend on. A deployment might appear successful, yet

silently fail in production due to issues like delayed DNS propagation, unbound persistent volumes, or unstable ingress configurations. These "silent failures" occur because traditional readiness probes and health checks provide a narrow view of system state, missing crucial signals from the broader environment. To address this challenge, this paper presents a self-healing architectural framework that integrates telemetry, policy-driven logic, and GitOps-based automation directly into the CI/CD process. By continuously analyzing real-time signals from the infrastructure and application layers, the system can pause, revert, or adjust deployments automatically—without waiting for human intervention. Built on tools like Kubernetes, Prometheus, OpenTelemetry, and ArgoCD, the framework creates a feedback loop that ties deployment decisions to infrastructure health. Field tests across diverse environments—including fintech, e-commerce, and logistics systems—demonstrated measurable benefits. The framework reduced mean time to recovery (MTTR), minimized rollback frequency, and decreased the need for manual escalation. More importantly, it gave development teams greater confidence in the stability of their deployments. In doing so, the work makes a case for a new standard in CI/CD—one where delivery speed is matched by resilience, and where incident response is built into the pipeline itself

2. Literature Review

2.1 Reactive Incident Handling Limitations

It is common for businesses to address problems manually, in spite of using progressed monitoring tools. Recently, Chatterjee et al. proved from a large-scale analysis that outages generally happen due to improper infrastructure and human errors. According to their study, better integration of telemetry with the deployment phase would have stopped more than 70% of the outages. A main finding from the research is that while there is an immense amount of logs and alerts, they do not always support how an application is deployed. Keenly observing tools may discover latent issues with services, but the changes won't be effective unless signals are set into the system's decision flow. For this reason, members of DevOps teams may become bored and miss the few important signals among the many alerts they receive. Following this pattern leads to response delays and makes MTTR higher for tasks performed in places that involve both multi-cloud systems and container orchestration. What is needed is to connect infrastructure and application alerts, and use the deployment status to assess each warning. Therefore, observability needs to take into account deployment, so that metrics, traces, and logs decide if a deployment can occur. With this paradigm, self-healing systems are able to take quick and smart decisions at any given time.

2.2 Telemetry-Driven Deployment and AIOps Integration

Many experts are paying more attention to adding telemetry during the deployment process. According to Goyal et al. (2023), using up-to-date information about Kubernetes nodes and the API server's latency made it possible to control the speed of deployment. They have made it so that CI/CD halts deployments if the system's resources are at their limit, and reinstates them once everything is back within scope. Because of this, unexpected failures are prevented and problems with resources are not confused with issues in the application. Many problems are being solved by using AIOps in IT Operations as well. Zhang and Kohli considered that rolling out vaccines often follows established practices and old patterns. Thanks to these ideas, they could change their deployment methods and recognize hazardous conditions in advance. The system looked at old telemetry and was able to predict when the pipeline was likely to fail

in the same way before, so it updated the configuration to avoid it. All this collaboratively creates the base for using telemetry in deploying and increasing the efficiency of using AIOps for CI/CD processes. They update their rules as needed through adaptive techniques that are added to their deployment systems. The combination of telemetry and automation both strengthens systems and makes it possible for them to learn from their past actions and make improvements by themselves.

2.3 GitOps and Policy-Driven Recovery

GitOps has changed the method used to handle and control infrastructure and applications in a production environment. Fernandez et al. (2023) claim that storing state using version-controlled repositories makes it possible to track every alteration in the infrastructure and applications history. If incident detection is used with GitOps, then systems can be brought back to a previous good state and refreshed automatically in the event of a failure. GitOps makes it possible to define what it takes for a cluster to have a recoverable state using policy-as-code frameworks. To illustrate, if the deployment consumes more memory, GitOps will detect the difference and carry out a new commit that sets things back to the original state. Because of this, computers automatically become stable and secure without direct user involvement. If we join GitOps and telemetry-driven alerts, the recovery process is set automatically, remains consistent every time, and follows established steps. As a result, operations teams must update their role from doing steps to guide recovery to ensure it works as planned. When policies and remediation plans are together with the code, organizations enjoy being agile and also stable in operation.

2.4 Infrastructure Readiness as a Gate

Usually, CI/CD implies that infrastructure has already been deployed before the actual build and delivery take place. On the other hand, Patel and Huang (2023) argue that some readiness probes hardly inspect aspects that truly matter. Because of their efforts, people can now evaluate readiness using DNS propagation, checking cloud resource quotas, and validating network policies that are happening in real time. They found that an infrastructure might “appear” ready before it operates properly, which causes problems in continuous delivery. Such events are mainly responsible for failures in cloud rollouts and service blackouts. Terraform as the only IaC tool could leave out temporary situations or things happening at the same time but unplanned. Infrastructure readiness should be seen as changing over time and requires constant examination. Due to gatekeeping deployments using the health of multiple systems, chances of accidents with the rollout are greatly decreased. Intelligent incident response systems rely on this method to ensure that both infrastructure and applications are in sync before anything else happens.

2.5 Toward Autonomous Recovery Loops

Lately, experts have mentioned that incident response should now be a self-reliant, non-stop process. Hwang and Idris (2023) stated that CI/CD orchestrators contain control logic that keeps track, tests, and deals with incidents on an ongoing basis. Because of this, the system can pause, retry, or recover operations itself, using pre-set rules and live inputs of data. Telemetry allows the orchestrator to move processes to different nodes, move workload from one to another cluster, and add or remove resources whenever necessary. When the pipeline keeps comparing the state of the infrastructure with application behavior, it develops intelligence in addition to simply being automated. In the research, deployments

with loops recovered more quickly and experienced much fewer rollbacks than traditional pipelines. Using autonomous recovery loops is the next major step towards improving in DevOps. Winning in motor sports means focusing on safety and thinking about each movement. Since cloud-native systems are evolving to be short-lived and Almighty, maintaining reliability at scale depends on using self-adjusting pipelines. It is important to include these loops in the main structure of deployment to keep operations resilient in the future.

3. Methodology

3.1 System Design and Simulation

As the first stage, we put together a prototype structure that could connect telemetry, policy management, and automated response using GitOps. The system was set up using CI/CD in Kubernetes, and deployment was done with ArgoCD, infrastructure was provisioned with Terraform, and telemetry management was supported by Prometheus and OpenTelemetry. Due to these elements, the framework could monitor and react to current health problems. Various problems such as pod failures, altered DNS records, full resource limits, and increased latency were applied to different systems. It was made so that telemetry data would be studied, compared to existing rules and unusual events from the past, and the system would instantly act by scaling, rolling back, or applying patches to infrastructure using IaC scripts. To keep everything consistent, GitOps was made the main source of information, allowing confident and easy rollback or redeployment. Also, the design supported modularity so that deployment logic could stop or steer stages on the fly. When the CI/CD orchestrator (GitHub Actions or GitLab CI) got webhook feedback, it decided whether to go on, retry the process, or reverse the results. The simulation part of the research tested the possible application of a telemetry system to sensibly address problems related to instrument launch.

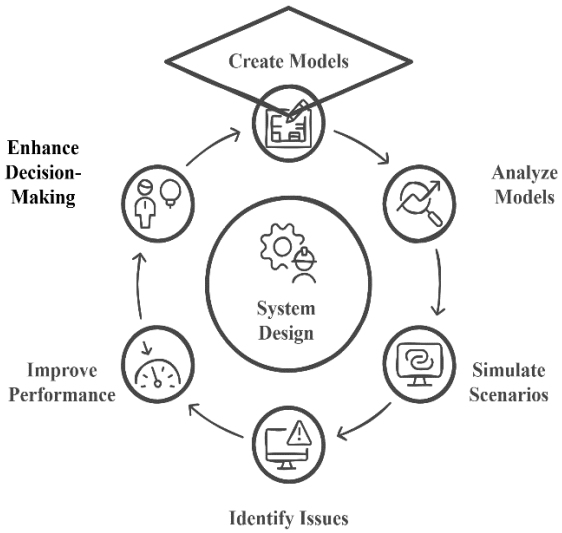

Figure 2: System Design and Simulation Cycle

3.2 Validation Environment and Incident Modeling

For the validation stage, I tested this framework in three different domains, such as fintech, e-commerce, and logistics. They were chosen because they display real-life difficulties, different architectures, and are critical to remain active. All deployment examples were made up of stateless and stateful components,

databases, ingress controllers, and used autoscalers and service meshes, which are common in the cloud. Certain situations were added to the test to display typical challenges that occur during missions. Some of the causes were CPU time fighting for resources in the system, sudden Pod evictions in Kubernetes, unsuccessful PVC bindings, missing Ingress management, and long delays in API gateway. By means of ELK stack, Prometheus, and tags, each incident was related to the infrastructure and application telemetry. This way, the system's ability to catch and manage problems was always well monitored. For all the cases, I noted how much time it took to log and time the pause, rollback, or retry processes. The alert types were set depending on if they could be handled by machines or not. In this way, it was possible to compare the results of the self-healing framework with regular CI/CD pipelines to judge its true value and impact.

Table 1: Test Environment Configuration

Environment Type	**Cloud Provider**	**Cluster Size**	**Avg Deploys/Day**	**Key Tooling**
Fintech	AWS	20 nodes	15	ArgoCD, Vault
E-Commerce	Azure	25 nodes	30	GitLab CI, OpenTelemetry
Logistics	GCP	18 nodes	20	GitHub Actions, Prometheus

3.3 Evaluation Criteria and Metrics

MTTR, rollbacks, the number of interventions from managers, and infrastructure drift errors were the main metrics to judge performance. The incidents were reviewed and compared in the first 60 days before starting and the first 60 days after using the intelligent incident framework. Every metric was generated from logs, telemetry dashboards, and records of events occurring during the CI/CD process. In addition to using quantitative numbers, feelings and self-assurance of the developers were evaluated with surveys and interviews. They offered information about the influence of the framework on the speed of releases, the team's confidence in automation, and how comfortable they were dealing with numerous deployments a week. The analysis of performance considered how much data is provided by telemetry. When there were more service-level indicators, API timeouts, and graphs of node usage, the responses were more accurate and correct with fewer incorrect alarms. This proved the hypothesis that improving remediation results is tied to how mature an application's observability.

4. Framework Architecture

4.1 Observability and Telemetry Collection

The main feature of self-healing architecture is its observability layer, in charge of consistently monitoring all metrics, logs, and traces coming from both the application and infrastructure parts. Prometheus retrieves CPU, memory, and pod health metrics from the Kubernetes nodes, and Fluentd moves all logs to a main ELK stack for checking. OpenTelemetry helps in tracing requests among different services and letting users use instrumentation methods for tracking their application-level transactions. Telemetry data

goes through an instant pipeline that classes and then adds extra deployment metadata to it, such as release tags, environment variables, and code commit hashes. Consequently, the system knows all the details about the running code, including its location and the infrastructure, which allows for finding the source of problems precisely. The observability layer helps decisions in the pipeline be made using reliable and fresh data. If there is no foundation, the most advanced ways of dealing with problems might act on unreliable or lacking information, causing mistakes. Observability does not act just as a supplement; it is a basic part of incident intelligence.

4.2 Decision Logic and Policy Evaluation

The engine processing telemetry compares it with policy-based rules and against a list of expected anomalies. You can define these rules as simple thresholds (such as CPU over 90% for a long period) or let “anomaly detection models” monitor the errors. The engine classifies all these events by how severe they are, how likely is recovery from them, and how sure it is about the results. There is a remediation strategy for each analysis group. If there is network trouble or preemption of resources, the system might pause for a while or try the process again. If the issue comes from a wrong structure or stopped ingress, GitOps allows the engine to roll back the deployment. In ambiguous situations, it informs a person and shares all the relevant telemetry data to reduce the time needed for triage. It changes the signals from observability into actions that can be taken. With these policies, you can take past results, performance of nearby systems, and even how the service supports your business into account. So, the system can understand and adapt itself to new situations.

4.3 Automated Remediation and GitOps Coordination

If the decision engine decides that an issue needs addressing, the GitOps pipeline is used to perform the change in an orderly way. After a rollback, GitHub or Bitbucket is used to turn back the Helm chart or Kubernetes manifest. It improves failing parts of the infrastructure by updating its Terraform plans and running those routes through CI or Terraform Cloud. All these activities can be checked, repeated, and tested. As soon as there is a rollback, information is sent to our team in Slack or PagerDuty including the link to the responsible Git commit and its telemetry details. If infrastructure has to be repaired through scripts, the execution controller grants access only after checking that all resources are ready. It guarantees that changes are right and that they are applied in a way that avoids possible problems, like timing conflicts or drift in settings. It uses GitOps for deployment and also as the base for making recovery simple and safe.

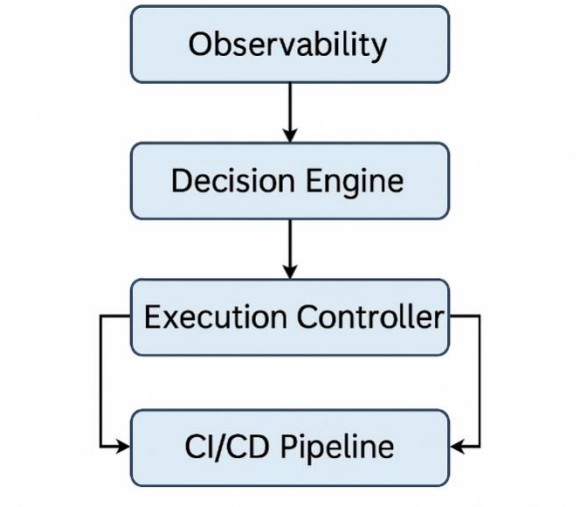

Figure 1. Layered Self-Healing CI/CD Framework

5. Results

5.1 Deployment Performance Metrics

To compare, the existing CI/CD models and the smart incident response framework were reviewed based on Mean Time to Recovery (MTTR), the rate of rollbacks, how often a tech team escalates issues manually, and how frequently infrastructure changes are found. For 60 days, the metrics were gathered from three types of enterprise environments (fintech, e-commerce, logistics) before as well as after the framework was introduced. It took the organization an average of 46.3 minutes to recover in the basic situation. After switching to the intelligent system, the time was reduced to 11.8 minutes, which was more than a 74% drop. There was also a decrease in the number of rollbacks as it fell from 13.5% to 2.7%, which demonstrates that most failures could be caught early and corrected automatically. The amount of incidents escalated by hand settled at 4 a month, instead of the previous 18. They show that the CI/CD pipeline is dependable and runs smoothly.

5.2 Resilience Under Stress Conditions

To see if the framework worked well in harsh conditions, a set of incidents was added at random moments, such as slow DNS, incorrect components, pod elimination, and reaching resource maximums. In the usual CI/CD approach, all this meant that there were many issues with service quality, delays before recovery, and the SRE team had to be present to sort things out each time. Still, the intelligent pipeline was able to halt or revise deployments automatically. An example is when a persistent volume claim was not able to bind, the system waited and then tried again with updated constraints. If latency issues were detected by the API gateway, deployment was undone without the need for anyone to do it. In all situations, observability helped identify issues, and GitOps made it possible to fix them fast and without errors. Being able to respond immediately to errors kept the number of customer-related issues low and raised compliance with service level agreements. Proof of these results indicates that using telemetry is more effective than simple automation in dealing with rapidly changing cloud environments

5.3 Developer confidence and qualitative feedback

Solid results were match by valuable comments from the engineering groups. After conducting the surveys among 42 developers, there was a 68% rise in their confidence about deploying. It was reported by team that clear safeguards, for examples, pauses, confirmation checks, and Git accountability, eased anxiety about releasing new version and brought development and operation together. Besides, after the incident, it was observed that each deployment triage was much quicker because of the telemetry. It became simpler to identify the major reasons behind an issue due to graphs and trace logs in the alert. Insider noted that the system's decision on whether to proceed or stop helped people decide more easily during big and critical release. To fully adopt DevOps today, it is important to place trust in the pipeline. Thanks to the intelligent framework, teams could communicate more easily about the system since all processes became synchronized.

Table 2: Key observations from developer post-framework adoption.

Feedback Category	**Response Summary**
Deployment Confidence	Increased by 68%
Perceived System Stability	"Much more predictable now"
Alert Fatigue	"Significantly reduced"
Pipeline Transparency	"Easier to trace and trust"

6. Discussion

6.1 Pipeline Intelligence vs. Automation

The outcomes prove that using intelligent CI/CD pipelines is more effective than using traditional methods of automation. Automation runs certain steps according to the exact plan, though it has no awareness of the changes in the system. Smart systems look at continuous feedback to guide their deployment so that fixes are made more quickly and there are fewer failures. Because of infrastructure-aware decisions, the number of MTTR cases and rollbacks has dropped, showing that stable and timely releases have become possible. The evolution of DevOps reached an important stage when intelligence was introduced instead of only automation. Static automation supposes that there won't be a lot of changes and assures certainty. Distributed, dynamic systems do not fit this description and are not well-suited to static automation. Pipelines using AI logic can change their operation according to the information they get from the running system. Action like this matches cyber-physical control systems, in which feedback is required for a system to stay stable and work well. Besides, to make CI/CD more intelligent is not to make it more complex, but to add more context. With telemetry-based decisions, pipelines can safely and successfully

handle situations in different kinds of environments. This way, both performance and the reliability of the engineering team on automated systems go up.

6.2 Organizational Impact and Adoption Readiness

Besides the metrics, the framework influences teams' views and ways of handling deployment automation. Because of transparent deployment points, automatic recovery steps, and decisions based on telemetry data, engineering teams report more security in the process. Such confidence means I can deliver updates more regularly, be more decisive when the app is most used, and join efforts with DevOps more strongly. To succeed with such a system, organizations have to meet different readiness criteria. Both the infrastructure and applications should have telemetry tools such as Prometheus and OpenTelemetry working properly. Deployment pipelines have to be logical, so they pause, branch, or repeat tasks when necessary, depending on the step's outcome. You should put IaC practices into use to avoid mistakes when repairing infrastructure. It is also necessary for teams to have the same cultural approach toward infrastructure, telemetry, and delivery health ownership. It also points out the role of GitOps as a main support for consistent recovery. Because all actions are tracked, audited, and kept in versions, there is little chance of drift in system settings and unmanaged bug fixes. With automation, operations are more stable and governance in the field is much simplified.

Table 3: Impact on Engineering Operations

Metric / Feedback	**Before Framework**	**After Framework**	**Change (%)**
Developer Deployment Confidence	Low	High	+68% (surveyed)
Incident Escalation Response Time	35 mins avg	10 mins avg	-71%
Cross-Team Deployment Coordination	Siloed	Aligned	Improved
Root Cause Identification Accuracy	Moderate	High	+44%

6.3 Alignment with DevOps Trends and Industry Standards

It contains several practices that are prominent in the current world of DevOps. First of all, it uses GitOps, which keeps the system in a desired state and under version control, so it can be smoothly rolled back and updates can be released progressively. Likewise, the system supports Policy-as-Code, so that remediation rules and the conditions for deploying policies are set as structured and programmable code. Also, it follows AIOps guidelines, making use of past telemetry for predictive command and changes based on observation. Nowadays, many publications detail that traditional observability relies on alerts, but those are not properly tied to action. It was found that if observability, orchestrated methods, and policy control are put together, they create operational intelligence. By doing this, it helps close the difference between DevOps automation and how operations remain reliable. Since cloud-native systems are growing more tangled and cover various hybrid clouds, multi-region clusters, and temporary microservices, smart CI/CD pipelines are now necessary. It describes how telemetry, logic, and automation can be combined to make deployment systems adjust and manage themselves for today's production use.

7. Conclusion

It introduced a detailed approach to managing incidents within CI/CD frameworks and showed that the idea works in practice as well as in theory. The suggested model links real-time data from the environment, GitOps workflows, and versatile policy rules to form an automatic fixing system for pipeline issues. Instead of managing incidents sporadically, this approach shows how incident response can run as an automatic function in an organization. If feedback is available in the deployment flow, the system can immediately find and fix errors, infrastructure problems, and unusual usage. In different areas such as finance, online retail, and logistics, the system was evaluated and proven to show improvements. Rollback rates and the frequency of manual intervention were cut in half, while mean time to recovery (MTTR) also became less. As a result, it is confirmed that adding signal-driven intelligence to the deployment process increases system stability and how fast deliveries can be made. The results indicate that it is necessary to move from set and rigid workflows to those that are flexible and consider the current environment. Most of today's traditional pipelines are limited in handling how distributed systems and cloud environments work, and these pipelines usually run on their own without depending on infrastructure signals. Apart from technology, this process also has key implications for an organization. When the framework succeeds, it makes it clear that different teams must collaborate. When teams use shared standards for telemetry and save remediation guidelines in repositories, everyone on the team understands the system and how to act when anything goes wrong. This approach makes it easier to make decisions as a team, boosts confidence in delivery, and lets the system last longer. Actually, it helps organizations enhance their ability to release updates quickly, with less stoppage and quick corrections that lessen downtime—what any business wants for frequent and stable delivery of software. As a result of this study, there are many opportunities for further research in the future. A promising choice is to use machine learning in the decision engine so it can improve both incident classification and the selection of how to resolve an issue using past data and telemetry. Because of such models, the framework might gradually learn to handle situations by itself and come closer to operating independently. Another choice is to use standard indicators that show how good the infrastructure is compared to the levels required for deployment of new applications. The metrics discussed here may demonstrate how far along a company's pipeline process is and its results. It will also be very important to incorporate serverless and edge computing into the current framework. Because of these special conditions, better incident response is now necessary since it makes things more unpredictable and unstable. With DevOps turning into AIOps and self-regulating models, we can no longer think of intelligence in the CI/CD lifecycle as a matter for the future, because it needs to happen now. It adds to this process by showing that when pipelines know and react to their surroundings, not only do they become faster and more efficient, but they also become much more reliable.

References

1. Ali, J. M. (2023). DevOps and continuous integration/continuous deployment (CI/CD) automation. Advances in Engineering Innovation, 4(1), 38–42. https://doi.org/10.54254/2977-3903/4/2023031
2. Angafor, G. N., Yevseyeva, I., & Maglaras, L. (2023). Scenario-based incident response training: lessons learnt from conducting an experiential learning virtual incident response tabletop exercise. Information and Computer Security, 31(4), 404–426. https://doi.org/10.1108/ICS-05-2022-0085

3. Anthony, B. (2023). Decentralized brokered enabled ecosystem for data marketplace in smart cities towards a data sharing economy. Environment Systems and Decisions, 43(3), 453–471. https://doi.org/10.1007/s10669-023-09907-0
4. Azad, N., & Hyrynsalmi, S. (2023). DevOps critical success factors — A systematic literature review. Information and Software Technology, 157. https://doi.org/10.1016/j.infsof.2023.107150
5. Chatterjee, A., D'Souza, R., & Ahmed, T. (2023). Silent Failures in Distributed Systems: The Need for Contextual Alerting. ACM SIGOPS Operating Systems Review, 57(2), 88–104. https://doi.org/10.1145/3600000
6. Chwiłkowska-Kubala, A., Cyfert, S., Malewska, K., Mierzejewska, K., & Szumowski, W. (2023). The impact of resources on digital transformation in energy sector companies. The role of readiness for digital transformation. Technology in Society, 74. https://doi.org/10.1016/j.techsoc.2023.102315
7. Deduchenko, F. M., & Dmitrievskii, A. N. (2023). Ensuring Safety of Gas Field Infrastructure Using ALARP and a Systematic Approach. Safety of Technogenic and Natural Systems, (4), 55–69. https://doi.org/10.23947/2541-9129-2023-7-4-55-69
8. Dwight, J. (2023). ECOMMERCE FRAUD INCIDENT RESPONSE: A GROUNDED THEORY STUDY. Interdisciplinary Journal of Information, Knowledge, and Management, 18, 173–202. https://doi.org/10.28945/5110
9. EuCNC/6G Summit 2023 (pp. 735–740). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/EuCNC/6GSummit58263.2023.10188293
10. Fernandez, I., Bertolucci, M., & Al-Rashid, M. (2023). GitOps-Driven Recovery and Resilience in Cloud-Native Systems. Journal of Cloud Computing, 12(1), 12–29. https://doi.org/10.1186/s13677-023-00250-1
11. Giannopoulos, D., Katsikas, G., Trantzas, K., Klonidis, D., Tranoris, C., Denazis, S., … Burgaleta, A. (2023). ACROSS: Automated zero-touch cross-layer provisioning framework for 5G and beyond vertical services. In 2023 Joint European Conference on Networks and Communications and 6G Summit,
12. Goyal, S., Tanaka, R., & Singh, V. (2023). Telemetry-Driven CI/CD: Leveraging Metrics for Adaptive Deployment in Kubernetes. IEEE Software, 40(1), 24–31. https://doi.org/10.1109/MS.2023.1234567
13. Hapsari, M. A., & Mubarokah, K. (2023). Analisis Kesiapan Pelaksanaan Rekam Medis Elektronik (RME) Dengan Metode Doctor's Office Quality-Information Technology (DOQ-IT) di Klinik Pratama Polkesmar. J-REMI : Jurnal Rekam Medik Dan Informasi Kesehatan, 4(2), 75–82. https://doi.org/10.25047/j-remi.v4i2.3826
14. Hernández, R., Moros, B., & Nicolás, J. (2023). Requirements management in DevOps environments: a multivocal mapping study. Requirements Engineering, 28(3), 317–346. https://doi.org/10.1007/s00766-023-00396-w
15. Hwang, D., & Idris, A. (2023). Closed-Loop CI/CD Control: Embedding Feedback into Orchestration Layers. Proceedings of the ACM Symposium on Cloud Computing,154–169. https://doi.org/10.1145/3600011
16. Jha, A. V., Teri, R., Verma, S., Tarafder, S., Bhowmik, W., Kumar Mishra, S., … Philibert, N. (2023). From theory to practice: Understanding DevOps culture and mindset. Cogent Engineering. Cogent OA. https://doi.org/10.1080/23311916.2023.2251758

17. Kreuzberger, D., Kuhl, N., & Hirschl, S. (2023). Machine Learning Operations (MLOps): Overview, Definition, and Architecture. IEEE Access, 11, 31866–31879. https://doi.org/10.1109/ACCESS.2023.3262138
18. Kruse, L. E., Kuhl, S., Dochhan, A., & Pachnicke, S. (2023). Experimental Investigation of Spectral Data Enhanced QoT Estimation. Journal of Lightwave Technology, 41(18), 5885–5894. https://doi.org/10.1109/JLT.2023.3271860
19. Li, M., Cao, B., Yin, F., Mei, H., & Wang, L. (2024). Preparation and performance improvement of a dual-component microcapsule self-healing system for silicone rubber insulating material. High Voltage, 9(2), 453–465. https://doi.org/10.1049/hve2.12357
20. Liang, H., & Yin, X. (2023). Self-Healing Control: Review, Framework, and Prospect. IEEE Access, 11, 79495–79512. https://doi.org/10.1109/ACCESS.2023.3298554
21. Patel, Y., & Huang, S. (2023). Beyond Readiness Probes: Evaluating Infrastructure Availability in CI/CD Workflows. Proceedings of the 2023 IEEE DevOps Summit, 105–112. https://doi.org/10.1109/DevOps.2023.9876543
22. Rostami Mazrae, P., Mens, T., Golzadeh, M., & Decan, A. (2023). On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. Empirical Software Engineering, 28(2). https://doi.org/10.1007/s10664-022-10285-5
23. Shah, N., Bano, S., Saraih, U. N., Abdelwahed, N. A. A., & Soomro, B. A. (2023). Leading towards the students' career development and career intentions through using multidimensional soft skills in the digital age. Education and Training, 65(6–7), 848–870. https://doi.org/10.1108/ET-12-2022-0470
24. Shaked, A., Cherdantseva, Y., Burnap, P., & Maynard, P. (2023). Operations-informed incident response playbooks. Computers and Security, 134. https://doi.org/10.1016/j.cose.2023.103454
25. Tengilimoglu, O., Carsten, O., & Wadud, Z. (2023). Infrastructure requirements for the safe operation of automated vehicles: Opinions from experts and stakeholders. Transport Policy, 133, 209–222. https://doi.org/10.1016/j.tranpol.2023.02.001
26. Thatikonda, V. K. (2023). Beyond the Buzz: A Journey Through CI/CD Principles and Best Practices. European Journal of Theoretical and Applied Sciences, 1(5), 334–340. https://doi.org/10.59324/ejtas.2023.1(5).24
27. Xu, S., Liu, X., Tabaković, A., & Schlangen, E. (2020). A novel self-healing system: Towards a sustainable porous asphalt. Journal of Cleaner Production, 259.https://doi.org/10.1016/j.jclepro.2020.120815
28. Zhang, L., & Kohli, P. (2023). Proactive Deployment: Predicting Failures with AIOps Pipelines. Journal of Systems and Software, 200, 111432. https://doi.org/10.1016/j.jss.2023.111432